

(12) **UK Patent Application** (19) **GB** (11) **2 340 264** (13) **A**

(43) Date of A Publication 16.02.2000

(21) Application No **9816482.5**

(22) Date of Filing **30.07.1998**

(71) Applicant(s)

International Business Machines Corporation
(Incorporated in USA - New York)
Armonk, New York 10504, United States of America

(72) Inventor(s)

Andrew John Smith
David Clark
Ian Holt

(74) Agent and/or Address for Service

C Boyce
IBM United Kingdom Limited, Intellectual Property
Dept, Hursley Park, WINCHESTER, Hampshire,
SO21 2JN, United Kingdom

(51) INT CL⁷

G06F 9/44 17/24

(52) UK CL (Edition R)

G4A AKS

(56) Documents Cited

EP 0435478 A2 **US 5101375 A**
Data Based Advisor Vol. 7, No. 7, July 1989, pages
104-108, and IAC Accession No. 07449594.

(58) Field of Search

UK CL (Edition Q) G4A AKS
INT CL⁶ G06F 3/023 9/44 17/24 17/25 17/26 17/30
Online: WPI, EDOC, INSPEC, COMPUTER

(54) Abstract Title

Filtering user input into data entry fields

(57) A user interface enables information to be entered by the user using entry fields, and the entered information to be modified using entry filters. Entry fields expose methods which allow their text to be queried, set and modified. The entry fields are adapted to allow for the addition of filter listeners, e.g. AutoComplete TextFilter, which implement methods for one or more user-generated entry field events, e.g. keyPressed, and, according to the constraints they apply to the text entered in the entry field, determine the final value of the entry field. By thus abstracting the notion of the entry filter rather than tightly integrating the filters with the implementation of the entry field itself, filtering is simplified and the filter may be readily applied to a plurality of disparate entry field implementations. Furthermore, multiple filters may simultaneously be applied to a single entry field in order to combine complementary filtering behaviours. Filters can also be applied to each of a plurality of editable text elements of a single entry field (fig. 6).

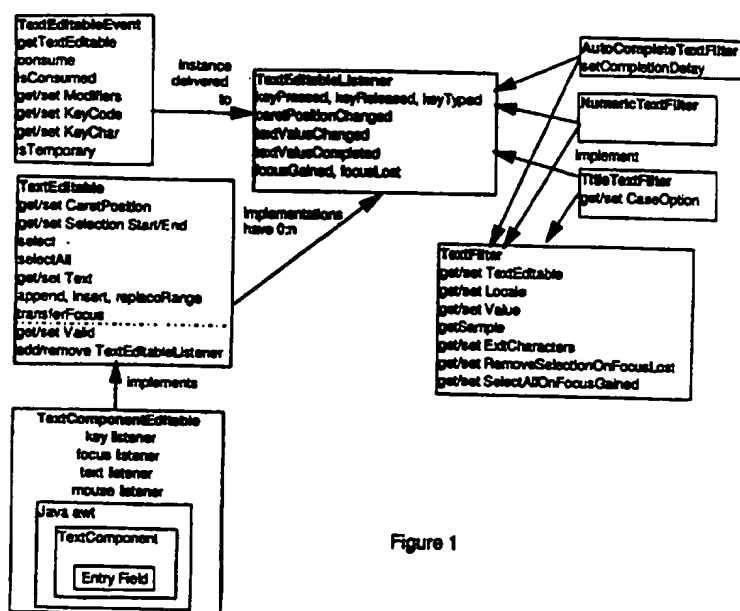


Figure 1

GB 2 340 264 A

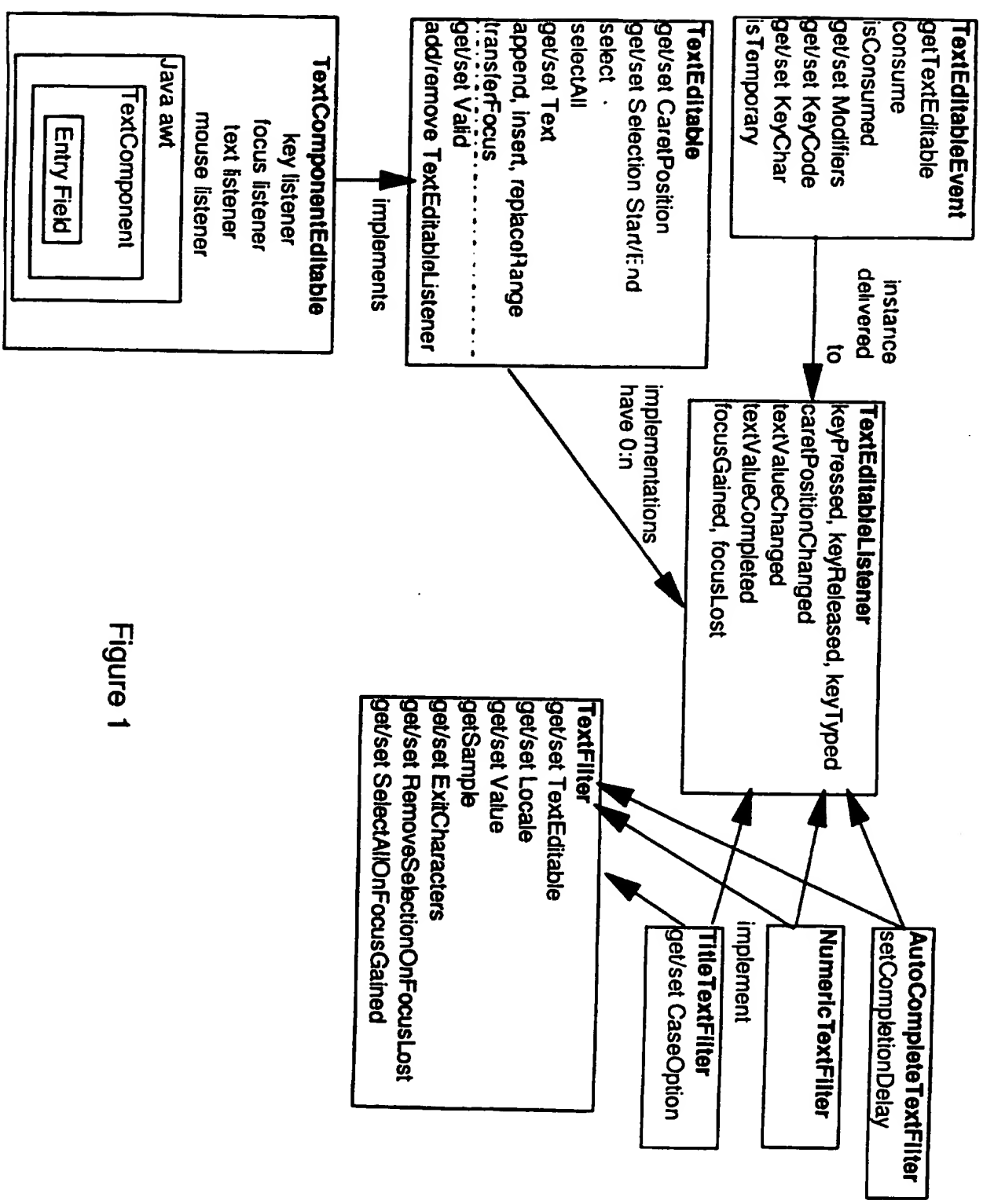


Figure 1

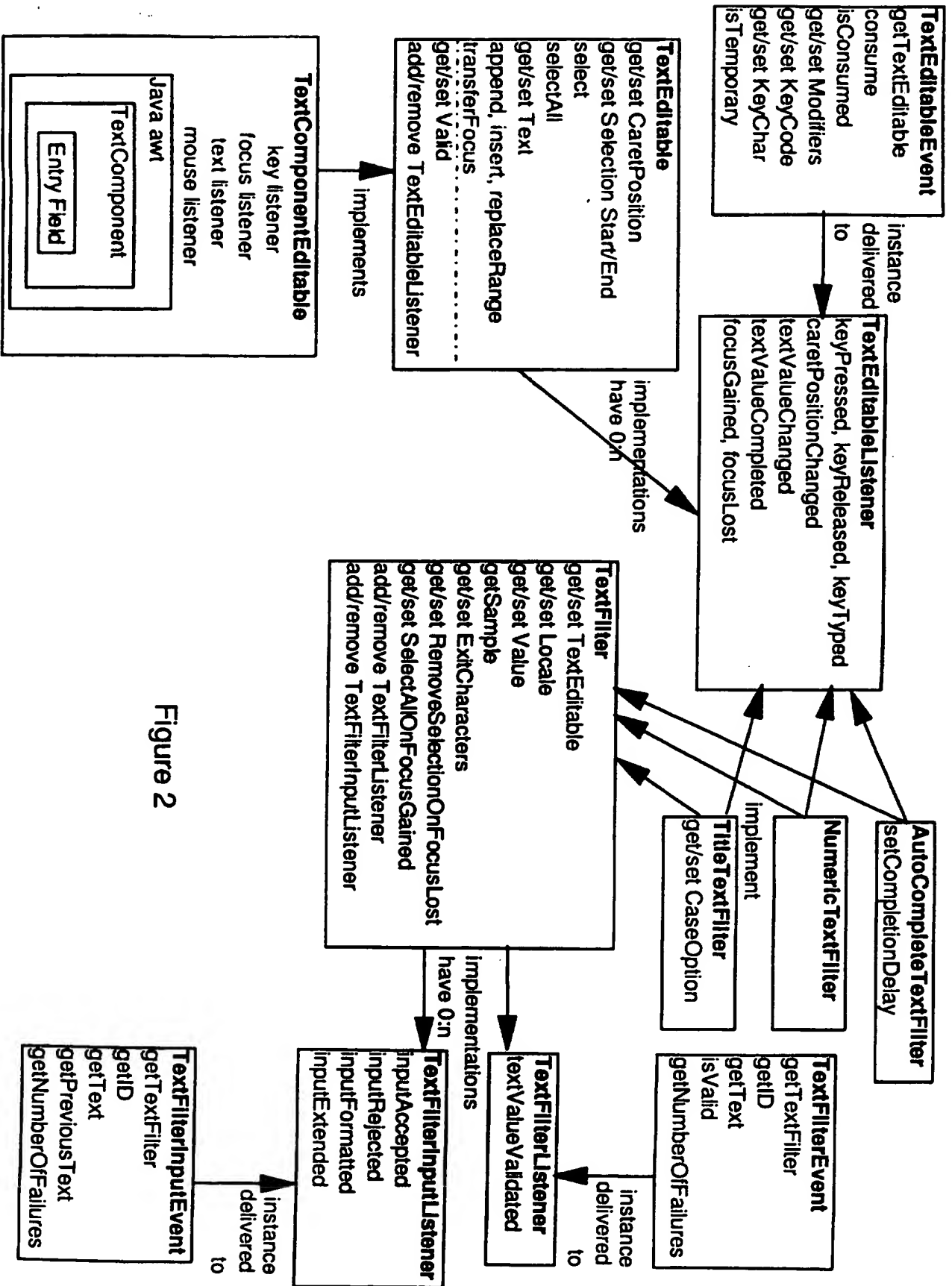


Figure 2

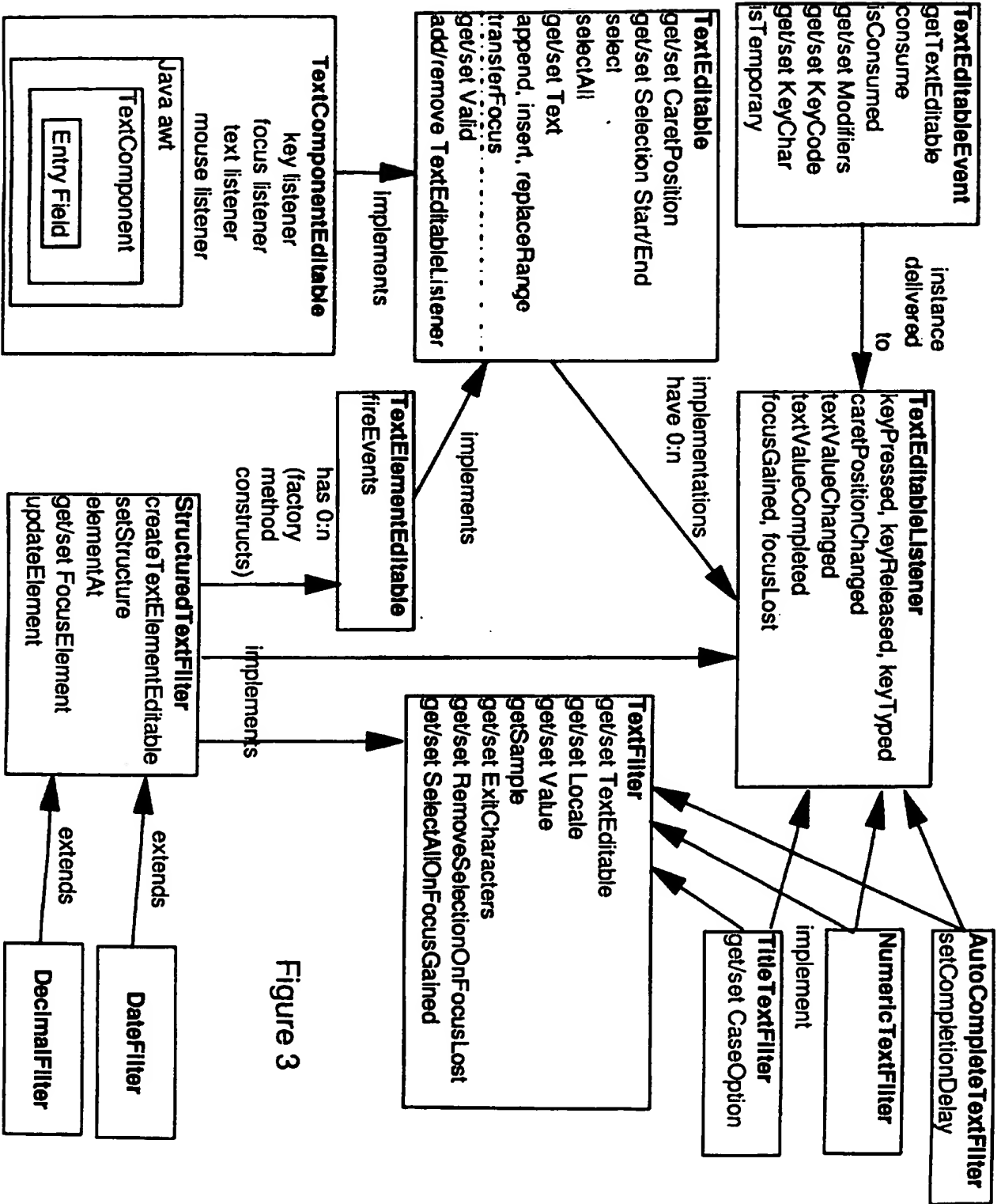


Figure 3

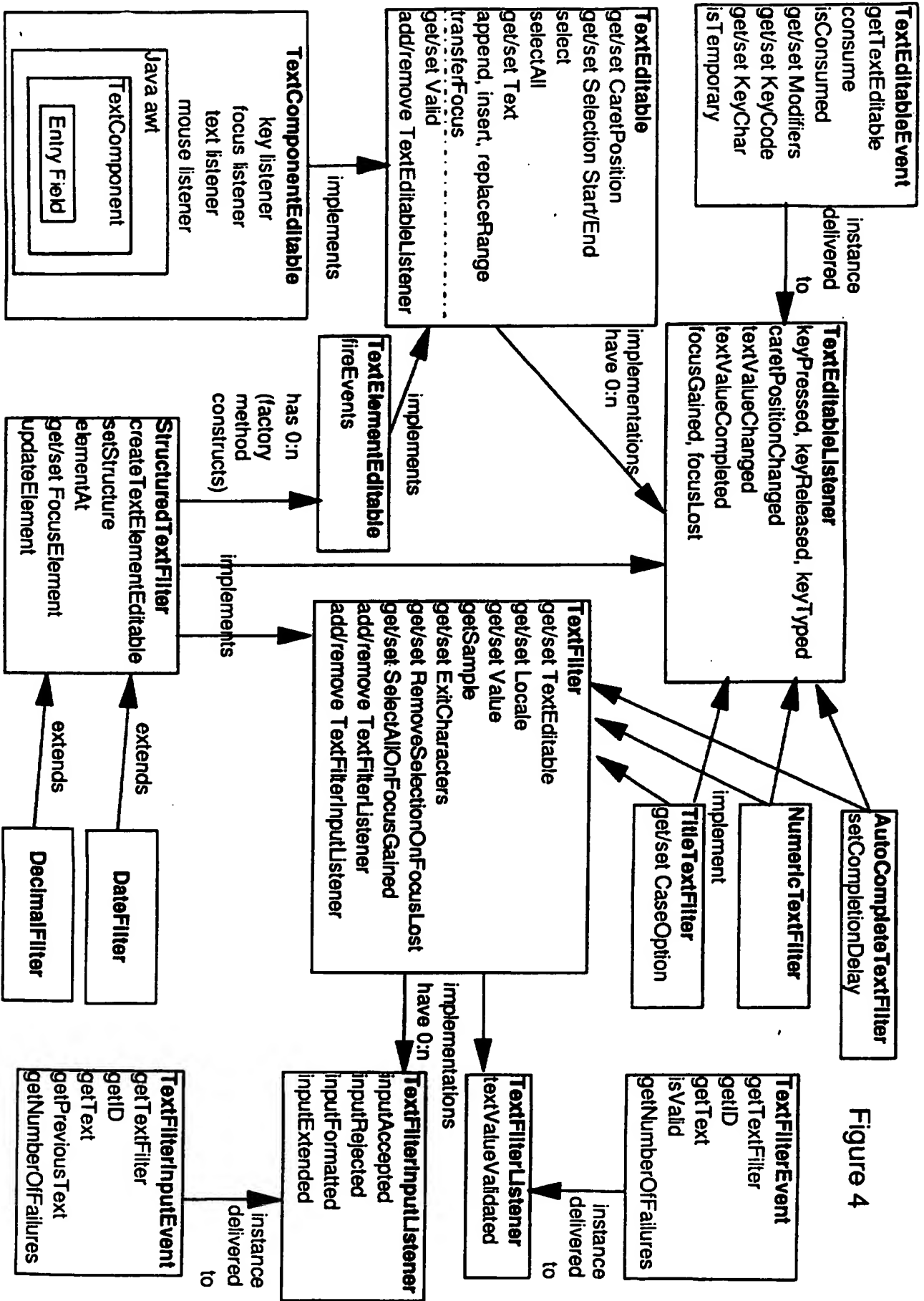


Figure 4

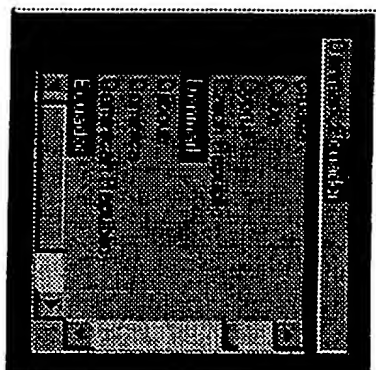


Figure 5(a)

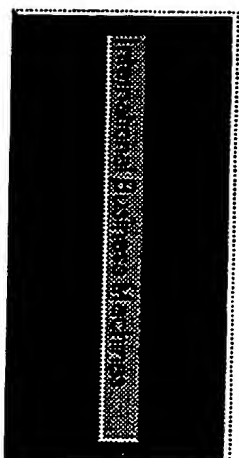
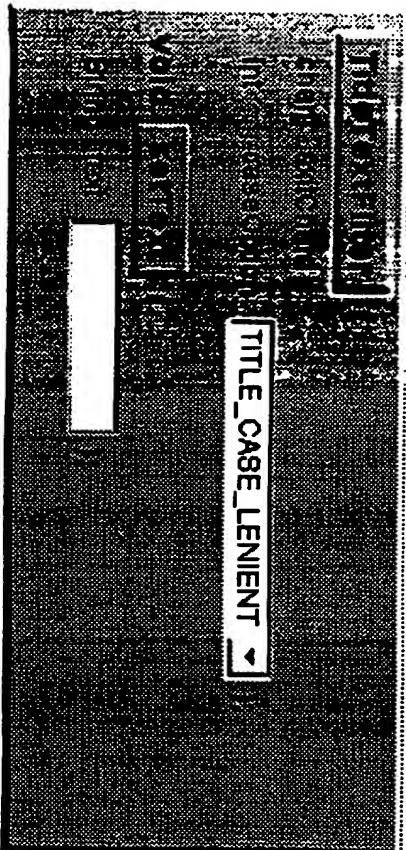


Figure 5(c)

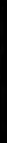


Figure 5(b)

DecimalFilter

int idna DecimalFilter.CURRENCY ▾

String locale US ▾

void setText ()

String loc

void setLocale ()

String locale (0)

void setDecimalFormat ()

String decimalFormat 0

0000.00

Figure 5(e)

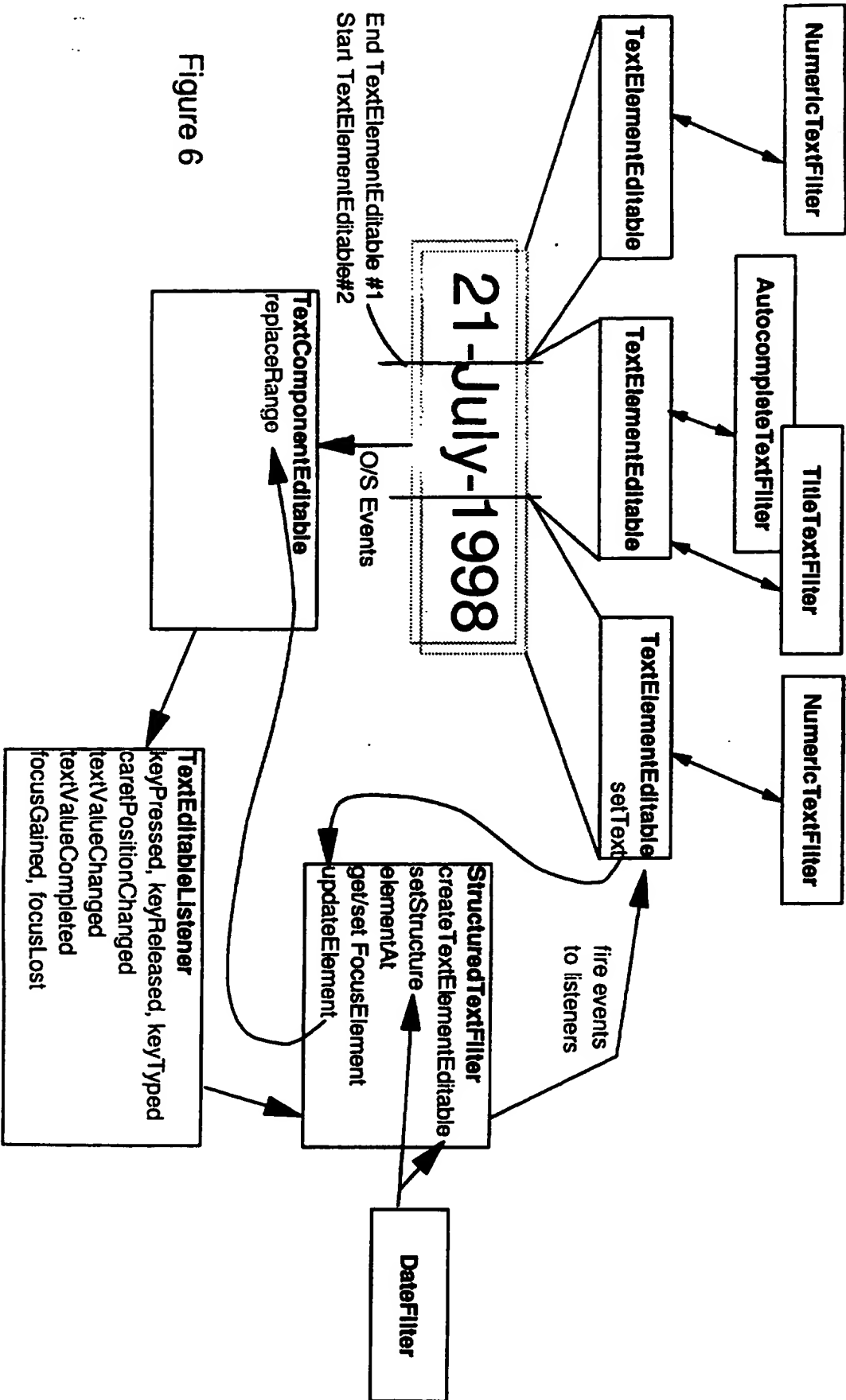


Figure 6

ENTRY FILTERS

The present invention relates to entry filters for entry field components of a user interface for a computer system.

5

A common requirement in a computer system user interface is entering information. This is typically done using entry fields. For text entry, typical characteristics of entry fields include a means for showing the insertion position, a means for moving the insertion position, and a means for entering text at the insertion position. Further characteristics may include the ability to select a section of the entered text, and operate on the selected text in a variety of ways.

10

There are certain constraints that can be enforced in order to get different types of data from the user. An example of this is ensuring the correct capitalization of a title. Another example is the restriction of entry to numeric characters only. A common way of achieving this is to extend the function of an entry field to monitor user input and impose the necessary constraints. This approach has several disadvantages. As user interfaces increase in diversity, and many variants of entry fields become available, the logic for applying constraints is not easily applied to these alternative entry fields. It may also be advantageous to combine constraint behaviours, which is not generally feasible when such behaviours are packaged with a particular entry field.

20

25

The present invention provides an entry filter according to claim 1.

30

The invention provides filters which encapsulate a set of rules or conditions and which can be combined for controlling or defining how a user's input is restricted or assisted. In the context of the invention, an entry field is anything allowing input from a plurality of input devices, and doesn't necessarily have to be textual.

35

40

The entry field cooperable with the entry filter according to the invention includes a means of querying and modifying the content of the entry field, and preferably of querying and modifying other characteristics of the entry field such as insertion point position, selection, etc. In addition, a means of monitoring changes to the content or other characteristics of the entry field is provided, along with a means of receiving notification of user input events and a means of modifying or suppressing those events.

Preferably, the entry filter includes means for specifying the entry field to which an entry filter is associated. This allows one or more filters to be associated with an entry field, although when more than one filter is associated with an entry field it is important to ensure that the sets of constraints do not adversely interfere with each other.

Preferably, the entry filter also includes means for receiving notification of significant events, such as the acceptance, rejection and modification of user input.

It should be noted that while entry filters according to the invention encapsulate the rules for a particular set of constraints, they are independent of any particular entry field implementation.

Embodiments of the invention will now be described with reference to the accompanying drawings, in which:

Figure 1 is a schematic diagram of the classes required to implement entry filters according to a preferred embodiment of the invention;

Figure 2 extends the diagram of Figure 1 to illustrate the classes required to enable external objects to communicate with the entry filters of Figure 1;

Figure 3 illustrates the classes of Figure 1 adapted to allow entry filters to work with a subset of an editable area managed by a structured entry filter;

Figure 4 extends the diagram of Figure 3 to illustrate the classes required to enable external objects to communicate with both the entry filters and structured entry filters of Figure 3;

Figures 5(a) to (e) illustrate some entry fields operating with entry filters and structured entry filters according to the invention; and

Figure 6 is a block diagram showing the components operating in an example of a structured entry filter.

The present invention according to a preferred embodiment is implemented as a software component for controlling the operation of a computer system on which it is run. The software component may be modified to run

on many different types of computer and operating system platforms. The computer program code according to the preferred embodiment is written in the Java programming language (Java is a trademark of Sun Microsystems Inc.). As is known in the art, the Java language provides for the abstract definition of required behaviours by a construct known as an interface. Concrete implementation of these behaviours can then be provided by classes which implement these interfaces.

In general terms, entry fields expose methods which allow their text to be queried, set and modified, for example, `getText`, `setText`, `append`, `insert`. Entry fields also generate events, for example, `keyPressed` or `textValueCompleted`. (In the present embodiment, these events are in fact method calls issued by the entry field to listeners which the entry field knows implement such methods.) The present invention operates by adapting an entry field to allow for the addition of listeners which implement methods for one or more entry field events and, according to the constraints they apply to the text entered in the entry field, determine the final value of the entry field. It is possible to associate one or more entry filters with an entry field, so that the desired composite behaviour can be obtained.

Examples of such listeners are an `AutoCompleteTextFilter`, `NumericTextFilter` and `TitleTextFilter`. The functionality of these filters is well known and computer users will be used to filling in entry fields having such functionality in on-screen forms. However, this functionality is commonly implemented as an integral part of an entry field, rather than as a plug-in behaviour as in the case of the present invention. This plug-in behaviour not only enables the same entry field implementation to be used for many individual types of behaviour, but also allows more than one entry filter to be applied to a field to generate complex behaviour. For example, an `AutoCompleteTextFilter` and a `TitleTextFilter` according to the invention can be applied to an entry field so that when a candidate is automatically returned by the `AutoComplete` filter to complete the field, it is then capitalised in a manner determined by the `TitleTextFilter`.

Referring now to Figure 1, a number of classes are shown, illustrating the operation of the preferred embodiment:

TextEditable

An encapsulated definition of the services expected of a text entry field is embodied in a `TextEditable` interface. Each characteristic required of a text entry field is thus supported by one or more methods defined by this interface. As is known in the art, the Java Abstract Windowing Toolkit (AWT) includes an entry field component within the `java.awt.TextComponent` class. Many of the methods defined by the `TextEditable` interface are also found on the `java.awt.TextComponent` class. A concrete implementation of the `TextEditable` interface is implemented within the `TextComponentEditable` class which is a simple wrapper for `java.awt.TextComponent`. A separate implementation `TextElementEditable`, Figure 3, allows filters to work with a sub-set of an editable area managed by a structured text filter, whose operation will be explained later.

The `TextEditable` interface defines the following methods which, unless otherwise noted, correspond to the methods implemented by `java.awt.TextComponent`:

`getCaretPosition` and `setCaretPosition` allow the text insertion position to be queried and modified.

`getSelectionStart`, `setSelectionStart`, `getSelectionEnd` and `setSelectionEnd` allow the start and end positions of the currently selected text to be queried and modified individually.

`select` allows the start and end positions of the currently selected text to be modified together.

`selectAll` selects the entire text.

`getText` and `setText` allow the current text to be queried and modified.

`append`, `insert` and `replaceRange` allow the current text to be manipulated in various ways.

`transferFocus` allows user input focus to be transferred to the next available component.

`getValid` and `setValid`, not in `TextComponent`, control whether the current text is to be shown as valid according to any associated constraints.

addTextEditableListener and removeTextEditableListener, not in
 TextComponent, allow the addition and removal of filters which are
 listeners for TextEditable events. Such filters must implement a
 TextEditableListener interface, described below, which describes the
 events which may be generated by implementations of TextEditable.

TextEditableListener

This interface describes the events which may be generated by
 implementations of TextEditable. It defines the following methods:

keyPressed, keyReleased and keyTyped allow keyboard events to be
 intercepted and any necessary actions to be taken. Using the methods on
 the TextEditableEvent class, described below, the details of the event
 may be modified, or the event suppressed.

caretPositionChanged is called when the text insertion point's position
 changes.

textValueChanged is called when the value in the text field changes.

textValueCompleted is called when the user has indicated that data entry
 is complete.

focusGained and focusLost are called when the text field gains or loses
 user input focus.

TextEditableEvent

An instance of TextEditableEvent is delivered to each of the methods on
 TextEditableListener. It provides information about the event, and a
 means of modifying or suppressing certain of those events. It defines the
 following methods:

getTextEditable returns a reference to the instance of TextEditable that
 generated the event.

consume allows the event to be suppressed. Subsequent listeners will
 still receive the event but can determine that it has been suppressed.

isConsumed returns whether or not the event has been suppressed.

getModifiers and setModifiers allow modifiers to be queried and modified. For keyboard events, these modifiers indicate which of the keyboard augmentation keys was active when the event was generated, as defined in the java.awt.event.KeyEvent class.

5

getKeyCode and setKeyCode allow key codes to be queried and modified. For keyboard events, these key codes indicate exactly which key generated the event, as defined in the java.awt.event.KeyEvent class.

10

getKeyChar and setKeyChar allow key characters to be queried and modified. For keyboard events, these key characters indicate which character, if any, corresponds to the key which generated the event.

15

isTemporary allows the focus transfer state to be queried, to determine whether it has permanently or temporarily transferred. Temporary focus loss occurs when the window that contains the text field loses focus.

TextFilter

20

An encapsulated definition of the services provided by a text entry filter is embodied in the TextFilter interface. Each characteristic of a text entry filter is thus supported by one or more methods defined by this interface. TitleTextFilter, AutoCompleteTextFilter and NumericTextFilter are concrete implementations of the TextFilter

25

interface.

The TextFilter interface defines the following methods:

30

setTextEditable allows the text filter to be associated with a single implementation of TextEditable. One or more text filters may be associated with a single implementation of TextEditable, although when more than one text filter is associated with a TextEditable it is important to ensure that the sets of constraints do not adversely interfere with each other.

35

getTextEditable returns a reference to the currently associated implementation of TextEditable.

40

getLocale and setLocale allow the java.util.Locale currently used by the text filter to be queried and modified. Implementations of the TextFilter

interface can use this `java.util.Locale` to assist with the provision of internationalisation support.

`getValue` and `setValue` allow the text in the associated `TextEditable` to be queried and modified using a Java class that is appropriate to the function of the text filter.

`getSample` returns a sample of valid entry for this text filter. This embodiment is able to use this to provide user assistance for data entry.

`getExitCharacters` and `setExitCharacters` allow the characters which will trigger focus transfer out of the associated `TextEditable` to be queried and modified.

`getRemoveSelectionOnFocusLost`, `setRemoveSelectionOnFocusLost`, `getSelectAllOnFocusGained` and `setSelectAllOnFocusGained` allow the focus behaviour of the filter to be queried and modified.

`addTextFilterListener`, `removeTextFilterListener`, `addTextFilterInputListener` and `removeTextFilterInputListener`, Figure 2, allow the addition and removal of listeners for `TextFilter` events. It will be seen that other objects in an application may want to benefit from the validation provided by text filters. In a database application, it may not be satisfactory for an object using information input by a user to rely upon the conventional `textValueCompleted` event to allow the object to receive the completed contents of an entry field, as this may anticipate the contents of the entry field being updated with a modified value reflecting the proper behaviour of the text filter.

Thus, in a preferred embodiment, Figure 2, two further interfaces `TextFilterListener` and `TextFilterInputListener` interfaces are provided. These interfaces described in more detail later, define events which may be generated by implementations of `TextFilter`, for example, `textValueValidated` tells a listener that the current value in the entry field, which may have been modified by the filter after completion by the user, is correct.

`TextComponentEditable`

This class is an example of a concrete implementation of the `TextEditable` interface. It enables a `java.awt.TextComponent` to be used with the other

classes in this embodiment. It primarily maps the methods defined by TextEditable to the equivalent methods on the java.awt.TextComponent class.

5 getValid and setValid have no equivalent on the java.awt.TextComponent class, and so are implemented by storing a Boolean data member.

10 TextComponentEditable, via the java.awt.TextComponent it encapsulates, listens to user generated events and as such acts as a key listener, a focus listener, and a text listener, in order that it can in turn issue the required TextEditable events, at the appropriate times. It is also a mouse listener, in order that it can monitor the caret position of the java.awt.TextComponent and issue the required caretPositionChanged events at the appropriate times. The textValueCompleted event is issued whenever
15 the enter key is pressed or the java.awt.TextComponent loses user input focus.

20 In use, an application instantiates a TextComponentEditable and calls the setTextEditable method on each of the previously instantiated filters. setTextEditable in turn calls addTextEditableListener on the TextComponentEditable to add the appropriate filter, for example TitleTextFilter, to the entry field. The filters then listen to any subsequent events and apply the appropriate constraints to the text that
25 may have been entered.

30 Filters such as NumericTextFilter will be inclined to wait until the textValueCompleted event before deciding whether a value entered is valid or needs modification, as such their implementation of events such as keyPressed, keyReleased or keyTyped will be minimal. AutoCompleteTextFilter on the other hand listens for every character
35 entered to determine if there is a match in its list of candidates for the text entered.

TextFilterListener

40 This interface describes the events which may be generated by implementations of TextFilter. It defines the following method:

40 textValueValidated is called whenever the text filter has attempted to establish whether or not the current text in the associated TextEditable is valid. This normally occurs when the associated TextEditable

indicates that data entry is complete by issuing the `textValueCompleted` event.

TextFilterEvent

5

An instance of `TextFilterEvent` is delivered to the method on `TextFilterListener`. It provides information about the event. It defines the following methods:

10

`getTextFilter` returns a reference to the instance of `TextFilter` that generated the event.

`getID` returns the unique identifier of the event.

15

`getText` returns the text which was in the associated `TextEditable` when the event was generated.

`isValid` returns whether or not the text in the associated `TextEditable` was found to be valid when the event was generated.

20

`getNumberOfFailures` returns the number of consecutive validation failures since the last successful validation.

TextFilterInputListener

25

A more detailed interface than `TextFilterListener`, this interface describes further events which may be generated by implementations of `TextFilter`. It defines the following methods:

30

`inputAccepted` is called when text entry has conformed to the constraints imposed by the text filter.

`inputRejected` is called when text entry has not conformed to the constraints imposed by the text filter.

35

`inputFormatted` is called when text entry is successfully modified to conform to a format required by the text filter.

40

`inputExtended` is called when the text entry is automatically added to by the text filter.

TextFilterInputEvent

Similar to TextFilterEvent, an instance of TextFilterInputEvent is delivered to each of the methods on TextFilterInputListener, and provides information about the event. It defines the following methods:

getTextFilter returns a reference to the instance of TextFilter that generated the event.

getID returns the unique identifier of the event.

getText returns the text which was placed into the associated TextEditable as the event was generated.

getPreviousText returns the text which was in the associated TextEditable before the event was generated. For inputFormatted and inputExtended events, the value returned will be different from that returned by getText.

getNumberOfFailures returns the number of input rejections since the last input that was accepted.

It will be seen that the invention could also be implemented with a single class combining TextFilterEvent and TextFilterInputEvent and/or another class combining TextFilterListener and TextFilterInputListener, although it is felt that developers employing the invention may prefer the added granularity provided by separate classes.

TitleTextFilter

This class is an example of a concrete implementation of the TextFilter interface, an example of which is shown on the right of Figure 5(c) while the main interface parameters are shown on the left of Figure 5(c). For more information on demonstrating user interface component operation in this fashion see British Patent Application No. 9713616.2 or US Application No. 08/037,605 (Docket No. UK9-97-043).

The purpose of TitleTextFilter is to ensure that text entered conforms to one of a set of defined capitalization schemes. In order to be able to intercept events from the associated TextEditable, it also implements the TextEditableListener interface.

TitleTextFilter provides the following additional methods:

getCaseOption and setCaseOption allow the capitalization scheme that is to be used by the filter to be queried and modified.

5

TitleTextFilter implements the methods of the TextFilter interface as follows:

10 setTextEditable adds the TitleTextFilter as a TextEditableListener to the supplied TextEditable, in order that the TextEditable events relevant to text capitalization are received. A reference to the supplied TextEditable is stored.

15 getTextEditable returns a reference to the currently associated TextEditable.

20 setLocale stores a reference to the supplied java.util.Locale. Locale services are used to perform text capitalization in the textValueChanged method described below.

25 getLocale returns a reference to the stored java.util.Locale.

30 getValue and setValue allow the current value to be queried and modified using the java.lang.String class. They are mapped directly to the getText and setText methods on the associated TextEditable.

35 getSample uses the currently selected capitalization scheme to return a sample of valid entry for this text filter.

40 setExitCharacters stores the supplied array of characters. These characters are used to trigger focus transfer in the keyPressed method described below.

45 getExitCharacters returns the stored array of exit characters.

50 setRemoveSelectionOnFocusLost and setSelectAllOnFocusGained store the supplied Boolean values. These values are used in the focusLost and focusGained methods described below.

55 getRemoveSelectionOnFocusLost and getSelectAllOnFocusGained return the stored Boolean values.

`addTextFilterListener` and `addTextFilterInputListener` add the supplied listener to the appropriate collection of listeners. These collections of listeners are used to issue appropriate events in the `textValueChanged`, `textValueCompleted`, and `keyTyped` methods described below.

5

`removeTextFilterListener` and `removeTextFilterInputListener` remove the supplied listener from the appropriate collection of listeners.

`TitleTextFilter` implements the methods of the `TextEditableListener` interface as follows:

10

`keyPressed` matches the key character that generated the event against the stored array of exit characters. If a match is found then focus is transferred by calling the `transferFocus` method on the associated `TextEditable`.

15

`keyTyped` uses the services provided by the stored `java.util.Locale` to determine whether the key character that generated the event is the first character of a new word. This information, in conjunction with the current capitalization scheme, is used to determine the correct case for the character. If the correct case differs from that of the key character, the key character is modified by using the `setKeyChar` method on the supplied `TextEditableEvent`, and an `inputFormatted` event is issued to the collection of `TextFilterInputListeners`. Finally, an `inputAccepted` event is issued to the collection of `TextFilterInputListeners`.

20

25

`textValueChanged` uses the services provided by the stored `java.util.Locale` to correctly capitalise the current text in the associated `TextEditable` using the current capitalization scheme, and an `inputFormatted` event is issued to the collection of `TextFilterInputListeners`.

30

`textValueCompleted` issues a `textValueValidated` event to the collection of `TextFilterListeners`.

35

`focusGained` uses the stored Boolean value set by `setSelectAllOnFocusGained` to determine whether the text in the associated `TextEditable` is to be selected. If it is, the `selectAll` method on the associated `TextEditable` is called.

40

focusLost uses the stored Boolean value set by
setRemoveSelectionOnFocusLost to determine whether the text in the
associated TextEditable is to be de-selected. If it is, the select method
on the associated TextEditable is called in such a way as to remove the
current selection without moving the text insertion point.

keyReleased and caretPositionChanged have empty implementations, as these
events are not relevant to text capitalization.

It will be seen that the implementation of TitleTextFilter does not
affect the content of the information in the entry field, only its
format. Other implementations of TextEditableListener,
AutoCompleteTextFilter and NumericTextFilter shown in Figures 5(a) and
5(b) respectively, define assistance for or a number of constraints on
the information entered.

AutoCompleteTextFilter, uses a supplied list of expected values to
provide prompted entry. It may also be provided with a list of separators
to enable the user to insert a list of entries in a single entry field.
In the case of Figure 5(a), this separator list comprises a semi-colon
and a slash.

AutoCompleteTextFilter also provides the method setCompletionDelay, which
allows other objects to specify the time in milliseconds after the last
user input before auto-completion is applied.

NumericTextFilter on the other hand needs to be provided with minimum and
maximum valid values, minimum and maximum lengths of numbers as well as
formatting parameters. Unlike the other two filters NumericTextFilter
does not need to expose any additional methods to other objects to allow
it to be configured appropriately.

In an enhancement of the present embodiment, entry filters according to
the invention work with a sub-set of an editable area managed by a
structured text filter, Figure 3. Each subset of an editable area is
implemented by a TextElementEditable class, which is an implementation of
TextEditable, adapted to work within a structured entry field.

TextElementEditable

TextElementEditable is a concrete implementation of the TextEditable interface. It is used in conjunction with StructuredTextFilter, described below, and represents an editable element in a complex structure. Instances of TextElementEditable are constructed by a factory method on StructuredTextFilter, and remain permanently linked to a single instance of StructuredTextFilter.

TextElementEditable stores the start position and the end position of the subset of the editable area which the element represents. It also stores the current text for this element, and a reference to the StructuredTextFilter to which it is linked.

This class implements the methods of the TextEditable interface as follows:

addTextEditableListener adds the supplied listener to the collection of listeners. This collection of listeners is used to issue appropriate events in the methods described below.

removeTextEditableListener removes the supplied listener from the collection of listeners.

setText updates the stored text for this element, and then calls an updateElement method on StructuredTextFilter, described below. It then issues a textValueChanged event to the collection of TextEditableListeners.

append, insert and replaceRange manipulate the current text in various ways, and call the setText method with the result.

getCaretPosition, setCaretPosition, getSelectionStart, setSelectionStart, getSelectionEnd, setSelectionEnd and select call the corresponding methods on the TextEditable associated with the StructuredTextFilter, adjusting the parameters and return values by using the stored start and end positions.

selectAll calls the corresponding method on the TextEditable associated with the StructuredTextFilter.

transferFocus attempts to locate the next element in the collection of active elements for the StructuredTextFilter and set the text insertion

point position of the associated `TextEditable` to the start position of that element, thus transferring focus to the next available element. If no next element is found, the `transferFocus` method on the associated `TextEditable` is called, thus transferring focus to the next available component.

StructuredTextFilter

This class is a concrete implementation of the `TextFilter` interface. It is used in conjunction with one or more `TextElementEditables` to provide a structured text entry mechanism. In order to be able to intercept events from the associated `TextEditable`, it also implements the `TextEditableListener` interface.

This class provides the following additional methods:

`createTextElementEditable` returns a new instance of `TextElementEditable` linked to the `StructuredTextFilter`.

`setStructure` defines the structure of editable and non-editable elements that will be managed by the `StructuredTextFilter`. This method accepts a text string to be used as the non-editable prefix, an array of `TextElementEditables` that will act as the editable elements of the structure, and an array of text strings to be used as the non-editable suffixes which follow each editable element. The length of the prefix, the length of the current text for each element, and the length of each suffix are used to compute the start position and the end position stored by each element.

`elementAt` returns the `TextElementEditable` which contains the supplied position. A `TextElementEditable` contains all positions from its own start position to the start position of the next `TextElementEditable`.

`setFocusElement` sets the text insertion point position of the associated `TextEditable` to the start position of the supplied `TextElementEditable`.

`getFocusElement` returns the `TextElementEditable` which contains the text insertion point of the associated `TextEditable`, or null if the text insertion point is within the non-editable prefix area.

updateElement uses the current text for the supplied TextElementEditable to re-compute the stored end position of the element, and also the stored start position and end position for each subsequent element. It then applies the element text to the associated TextEditable by calling its replaceRange method.

This class implements the methods of the TextFilter interface as follows:

setTextEditable adds the StructuredTextFilter as a TextEditableListener to the supplied TextEditable, in order that the TextEditable events relevant to structured text entry are received. A reference to the supplied TextEditable is stored.

getTextEditable returns a reference to the currently associated TextEditable.

setLocale stores a reference to the supplied java.util.Locale.

getLocale returns a reference to the stored java.util.Locale.

getValue and setValue allow the current value to be queried and modified using the java.lang.String class. They are mapped directly to the getText and setText methods on the associated TextEditable, in the expectation that this method will be overridden by classes extending StructuredTextFilter.

getSample returns an empty string in the expectation that this method will be overridden by classes extending StructuredTextFilter.

setExitCharacters stores the supplied array of characters. These characters are used to trigger focus transfer in the keyPressed method described below.

getExitCharacters returns the stored array of exit characters.

setRemoveSelectionOnFocusLost and setSelectAllOnFocusGained store the supplied Boolean values. These values are used in the focusLost and focusGained methods described below.

getRemoveSelectionOnFocusLost and getSelectAllOnFocusGained return the stored Boolean values.

`addTextFilterListener` and `addTextFilterInputListener` add the supplied listener to the appropriate collection of listeners. These collections of listeners are used to issue appropriate events in the `textValueChanged`, `textValueCompleted` and `keyTyped` methods described below.

5

`removeTextFilterListener` and `removeTextFilterInputListener` remove the supplied listener from the appropriate collection of listeners.

10

`StructuredTextFilter` implements the methods of the `TextEditableListener` interface as follows:

`keyPressed` matches the key character that generated the event against the stored array of exit characters. If a match is found then focus is transferred by calling the `transferFocus` method on the associated `TextEditable`. If no match is found, this method causes the `TextElementEditable` returned by the `getFocusElement` method to issue a `keyPressed` event to its collection of `TextEditableListeners`.

15

`keyTyped` and `keyReleased` cause the `TextElementEditable` returned by the `getFocusElement` method to issue a `keyTyped` or `keyReleased` event to its collection of `TextEditableListeners`.

20

`textValueChanged` parses the new text in the associated `TextEditable`, using the prefix and the suffixes as delimiters, to obtain the new text for each editable element. If the prefix or any of the suffixes does not appear fully in the new text, then they are restored by building the complete value using the prefix, the suffixes and the current text for each editable element, and applying that complete value to the associated `TextEditable` by calling the `setText` method. In this way the preservation of the non-editable prefix and all of the non-editable suffixes is ensured.

25

30

`textValueCompleted` issues a `textValueValidated` event to the collection of `TextFilterListeners`.

35

`focusGained` uses the stored Boolean value set by `setSelectAllOnFocusGained` to determine whether the text in the associated `TextEditable` is to be selected. If it is, the `selectAll` method on the associated `TextEditable` is called. If not, the `setFocusElement` method is called supplying the first `TextElementEditable` as the parameter.

40

focusLost uses the stored Boolean value set by
 setRemoveSelectionOnFocusLost to determine whether the text in the
 associated TextEditable is to be de-selected. If it is, the select method
 on the associated TextEditable is called in such a way as to remove the
 5 current selection without moving the text insertion point. Finally, it
 causes the TextElementEditable returned by the getFocusElement method to
 issue a focusLost event to its collection of TextEditableListeners.

caretPositionChanged determines whether the new text insertion point is
 10 contained within a different TextElementEditable to that which contains
 the previous text insertion point. If so, it causes the
 TextElementEditable which contains the previous text insertion point to
 issue a focusLost event to its collection of TextEditableListeners, and
 then causes the TextElementEditable which contains the new text insertion
 15 point to issue a focusGained event to its collection of
 TextEditableListeners.

DateFilter

20 This class is an example of an extension to the StructuredTextFilter
 class. Its purpose is to create and maintain a structure of editable
 elements and non-editable areas which together form a filter for date
 and/or time entry. An example is shown in Figure 5(d). It generally makes
 use of two of the concrete implementations of TextFilter, namely
 25 NumericTextFilter and AutoCompleteTextFilter described above.

NumericTextFilter, as described above, restricts input to numeric
 characters, and can also be configured with range and formatting
 information. It is used for numeric elements of the date format, such as
 30 day number, month number, year, hour, minute, and second.

AutoCompleteTextFilter is used for elements of the date format which have
 a small set of possible values, such as day name, month name, era, and
 time zone.

35 The structure for a DateFilter is specified by a pattern composed of
 masking characters as defined by the java.text.SimpleDateFormat class.
 Alternatively, the structure can be specified using one of the predefined
 formats defined by the java.text.DateFormat class. In this case, Locale
 40 services are used to convert each of these predefined formats to a
 pattern.

The pattern is used to determine the non-editable prefix, the requisite number of `TextElementEditables`, the appropriate text filters to be associated with each `TextElementEditable`, and the non-editable suffixes. Each required `TextElementEditable` is created using the

5 `createTextElementEditable` method, and the appropriate text filters are then created and associated with it. The prefix, the `TextElementEditables` and the suffixes are then supplied to the `setStructure` method of `StructuredTextFilter`.

10 `DateFilter` also provides the following additional methods:

`setDateUsed` and `setTimeUsed` store the supplied Boolean values. These values control the inclusion of the date and time parts of the structure.

15 `getDateUsed` and `getTimeUsed` return the stored Boolean values.

`setDateFormatStyle` and `setTimeFormatStyle` store the supplied predefined formats defined by `java.text.DateFormat`. These values control the format of the date and time parts of the structure.

20

`getDateFormatStyle` and `getTimeFormatStyle` return the stored predefined formats.

`setPattern` allows a custom pattern to be specified.

25

`getPattern` returns the current pattern.

`DateFilter` also overrides the following methods of `StructuredTextFilter`:

30

`setLocale` stores a reference to the supplied `java.util.Locale`. Locale services are used to apply the correct formatting characteristics. Changing the Locale may cause the entire structure to be recreated.

`getLocale` returns a reference to the stored `java.util.Locale`.

35

`setValue` allows the current value to be modified using the `java.util.Date` class. Locale services are used to convert the supplied `java.util.Date` value to a `java.lang.String` value, which is passed to the `setText` method on the associated `TextEditable`.

40

getValue uses Locale services to return a java.util.Date object representing the current value.

getSample uses Locale services to return a java.lang.String representing the current date formatted according to the current pattern.

Figure 6 shows an instantiation of structured text filter managing a date entry field. The date field in this case has a pattern comprising no prefix, two TextElementEditables each with a suffix of "-" and a third TextElementEditable without a suffix. The instance of the date entry field instantiates the TextElementEditables and passes the prefix, the TextElementEditables and the suffixes to an instance of StructuredTextFilter which manages user interaction with an instance of TextComponentEditable which is used to display the entry field on a screen and to listen to user generated events relating to the entry field. The instance of date entry field also instantiates and adds the appropriate filters to the respective TextElementEditables. The first having an associated NumericTextFilter with a minimum of 1 and a maximum of 31, the second with an associated AutoCompleteTextFilter having a list of candidates being the names of the months, and the third having an associated NumericTextFilter with a minimum and maximum determining the first of last year accepted. Also shown is a TitleTextFilter associated with the second element. This enables different capitalisation from that of entries in the list of candidates to be enforced.

When an event, for example a key press, occurs, TextComponentEditable issues the keyPressed event to its listeners. One of these listeners is the instance of StructuredTextFilter. StructuredTextFilter then needs to determine which of the TextElementEditables has user input in focus which it does by calling getFocusElement.

StructuredTextFilter then causes the TextElementEditable returned to issue a keyPressed event to each of its listeners. This is made possible because each TextElementEditable also implements a list of methods corresponding to the events it can generate. For example, a method fireKeyPressed is available to enable objects to cause a TextElementEditable to call keyPressed on each of its listeners. These listeners are the appropriate text filters for the element.

Depending on the event and the text filter, the text filter may need to change the contents of the entry field. It does this by calling setText

on its associated `TextEditable` - this will be the appropriate `TextElementEditable`. The `TextElementEditable` in turn calls `updateElement` on the `StructuredTextFilter`. `updateElement` then calls `replaceRange` on the associated `TextEditable` to substitute the value for the old value. The
 5 associated `TextEditable` will be the `TextComponentEditable` with which the user is interacting. `updateElement` then adjusts the start and end positions for each `TextElementEditable` before finishing.

The instance of `DateFilter` can determine the value of the entire entry
 10 field at any time by calling `getValue` on `StructuredTextFilter` which is in turn mapped onto `getText` on the associated `TextEditable`, which is the `TextComponentEditable`.

A further example of a structured text filter, `DecimalFilter`, is shown in
 15 Figure 5(e). `DecimalFilter` extends the `StructuredTextFilter` class to provide currency, percentage, and number entry behaviour. `DecimalFilter` constrains the user's input to be a number which can admit negative values and/or a decimal point, currency symbols, percentage, etc.

All decimal number formats provided by Java Development Kit 1.1
 20 internationalisation can be supported. The required format may be specified, or the locale default used.

Since these formats vary from country to country, the expected format is
 25 determined by the default or specified locale. `DecimalFilter` determines the type and order of the elements, as well as the appropriate element prefix and suffixes. Numeric text filters are applied to the elements representing the integer and fraction parts of the number. An appropriate prefix and/or suffixes are set to represent the currency symbol(s),
 30 percent symbol(s), decimal separator(s), and negative symbol(s). Locale services are used to obtain these symbols, and their correct positions.

For example, -123.45 as a currency in a US English locale would look like
 35 this: (\$123.45). This can be considered as two elements: the first element uses a numeric text filter, and has a suffix set to the decimal separator for the locale; the second element uses a numeric text filter constrained to two digits, and has a suffix of ")". The "(" and the currency symbol are combined to form the prefix.

It will be seen that, while the present embodiment is described in terms
 40 of Java, the invention is applicable to components written in other

languages. For example, the components could also be written as a set of ActiveX Controls possibly for use by a visual builder such as Microsoft Visual Studio '97. Such controls can be used to build an application in a similar manner to that for Java without departing from the scope of the invention.

5

It should also be noted that for simplicity and clarity, the description has in many places described the operation of the invention in terms of the names of classes themselves. It will of course be apparent to those skilled in the art, that it is in fact instances of these classes that operate at run-time, with those instances in general being given different names to those of the classes of which they are instances. This convention should not, however, detract from the overall level of the disclosure.

10

CLAIMS

- ✓ 1. An entry filter component cooperable with an entry field component, said entry field component being instantiable to receive user input, to
5 store said user input, to change status responsive to said user input, to generate one or more events indicative of said status and being adapted to allow instances of the entry field component to add one or more entry filter components as listeners to the or each event, said entry filter component being instantiable to respond to one or more of the or each
10 events, to read said user input and to modify said user input according to one or more conditions associated with said entry filter component.
2. An entry filter according to claim 1 wherein instances of said entry field are adapted to generate an event each time said user input
15 changes and wherein said entry filter is a prompted entry filter, instances of said prompted entry filter having an associated list of candidates and being adapted to respond to said user input change event by comparing said user input to entries in their associated list of candidates and, responsive to said user input matching an entry in their
20 associated list of candidates, modifying said user input to match said entry.
3. An entry filter according to claim 2 wherein instances of said prompted entry filter include a time delay characteristic determining a
25 length of time after a last user input change event before said user input is compared to entries in their associated list of candidates.
4. An entry filter according to claim 1 wherein instances of said entry field are adapted to generate an event when said user input is
30 complete and wherein said entry filter is a numeric filter, instances of said numeric filter having characteristics determining the range of said user input and being adapted to respond to said user input complete event by comparing said user input to said characteristics and, responsive to said user input lying outside the range determined by said
35 characteristics, modifying said user input to lie within said range.
5. An entry filter according to claim 1 wherein instances of said entry field are adapted to generate an event when said user input is
40 complete and wherein said entry filter is a title text filter, instances of said title text filter having a characteristic determining the character case of said user input and being adapted to respond to said

user input complete event by modifying said user input to conform to the character case.

5 6. An entry filter according to claim 1 wherein instances of said entry filter are adapted to receive an object from instances of said entry field when an event is generated by said instances of entry field, said instances of entry filter being adapted to use said object to control their response to said event.

10 7. An entry filter according to claim 6 wherein said instances of entry filter are adapted to use said object to generate a reference to the instance of the entry field that generated said event.

15 8. An entry filter according to claim 1 in which instances of said entry filter generate one or more events indicative of said status and are adapted to add one or more objects as listeners to the or each event.

20 9. An entry filter according to claim 8 in which said events include events indicative of the user input having being validated by said entry filter instance, the user input conforming to the conditions associated with the entry filter instance, the user input not conforming to the conditions associated with the entry filter instance, the user input having been successfully modified to conform to the conditions associated with the entry filter instance or the user input having been automatically added to by the entry filter instance.

25 10. An entry filter as claimed in claim 1 wherein said entry field is an entry field adaptor component instantiable to store a definition of a subset of an editable area within said instance of entry field and to store a value associated with said editable area, said entry field adaptor component being further adapted to allow instances of the entry field adaptor component to add one or more instances of entry filter components as listeners to the entry field adaptor component.

30 11. An entry filter as claimed in claim 1 wherein said entry filter is a structured entry filter manager component being instantiable to receive a pattern dividing an instance of an entry field into a plurality of editable areas, at least one of said editable areas having an associated instance of entry filter, instances of said filter manager being adapted to add themselves as listeners to events generated by an entry field instance and to relay said events to any entry filter instances

associated with an editable area of said entry field to which said event applies.

12. A set of components for facilitating user input in an application,
said set of components comprising:

an entry field component being instantiable to receive user input,
to store said user input, to change status responsive to said user input,
to generate one or more events indicative of said status and being
adapted to allow instances of the entry field component to add one or
more entry filter components as listeners to the or each event, and

one or more entry filter components being instantiable to respond
to one or more of the or each events, instances of the or each entry
filter being adapted to read said user input and to modify said user
input according to one or more conditions associated with the or each
entry filter component.

13. A set of components as claimed in claim 11 comprising:

a structured entry filter manager component being instantiable to
receive a pattern dividing an instance of an entry field into a plurality
of editable areas, at least one of said editable areas having an
associated instance of entry filter, instances of said filter manager
being adapted to add themselves as listeners to events generated by an
entry field instance and to relay said events to any entry filter
instances associated with an editable area of said entry field to which
said event applies; and

a structured entry filter component being instantiable to determine
said pattern according to a required format and to pass said pattern to
said structured entry filter manager.

14. A set of components as claimed in claim 13 wherein said pattern
includes a value to be used as the non-editable prefix and an array of
values to be used as non-editable suffixes within each editable area of
the entry field.

15. A set of components as claimed in claim 13 wherein said structured
entry filter component is a Date Filter, said Date Filter being
instantiable to store a date format pattern.

16. A set of components as claimed in claim 13 wherein said structured entry filter component is a Decimal Filter, said Decimal Filter being instantiable to store a decimal number format pattern.

5 17. A set of components as claimed in claim 13 further comprising:

an entry field adaptor component instantiable to store a definition of an editable area within an instance of entry field and to store a text value associated with said editable area, said entry field adaptor
10 component being further adapted to allow instances of the entry field adaptor component to add one or more instances of entry filter components as listeners to the entry field adaptor component.

18. A set of components as claimed in claim 17 wherein instances of
15 said structured entry filter component are adapted to add one or more instances of the or each entry filter to each entry field adaptor component instance according to said required format.

19. A set of components as claimed in claim 18 wherein instances of
20 said structured entry filter relay said events to an instance of entry field adaptor component associated with an editable area of said entry field to which said event applies, said entry field adaptor component instance being adapted to relay said event to any instance of entry filter component listening to said entry field adaptor component.

25 20. A set of components as claimed in claim 18 wherein instances of said entry field adaptor component relay modifications of user input from instances of entry filters listening to said entry field adaptor component instances to an associated structured entry filter manager
30 instance, said structured entry filter manager instance being adapted to modify the user input stored in said entry field instance.

21. A set of components as claimed in claim 18 wherein said structured entry filter manager component instance is adapted to return a reference
35 to the instance of entry field adaptor component representing the subset of the editable area of said entry field instance which contains a specified position.

22. A computer program product stored on a computer readable storage
40 medium for, when executed on a computer, facilitating user input in an

application, the product comprising an entry filter component as claimed in claim 1.



The
Patent
Office
28

Application No: GB 9816482.5
Claims searched: 1-22

Examiner: Melanie Gee
Date of search: 13 January 1999

Patents Act 1977
Search Report under Section 17

Databases searched:

UK Patent Office collections, including GB, EP, WO & US patent specifications, in:
UK CI (Ed.Q): G4A (AKS)
Int CI (Ed.6): G06F 3/023, 9/44, 17/24, 17/25, 17/26, 17/30
Other: Online: WPI, EDOC, INSPEC, COMPUTER

Documents considered to be relevant:

Category	Identity of document and relevant passage	Relevant to claims
A	EP 0435478 A2 (EMTEK HEALTH CARE SYSTEMS), see especially page 7 line 40 - page 10 line 44.	
A	US 5101375 A (GOLDHOR), see whole document.	
A	Data Based Advisor Vol. 7, No. 7, July 1989, C Currid et al., "From here to there (and back again!)", pages 104-108 (see discussion of "Data Junction" and also IAC Accession No. 07449594).	

X	Document indicating lack of novelty or inventive step	A	Document indicating technological background and/or state of the art.
Y	Document indicating lack of inventive step if combined with one or more other documents of same category.	P	Document published on or after the declared priority date but before the filing date of this invention.
&	Member of the same patent family	E	Patent document published on or after, but with priority date earlier than, the filing date of this application.

**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

☒ **BLACK BORDERS**

☒ **IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**

☒ **FADED TEXT OR DRAWING**

☐ **BLURRED OR ILLEGIBLE TEXT OR DRAWING**

☐ **SKEWED/SLANTED IMAGES**

☒ **COLOR OR BLACK AND WHITE PHOTOGRAPHS**

☒ **GRAY SCALE DOCUMENTS**

☐ **LINE(S) OR MARK(S) ON ORIGINAL DOCUMENT**

☐ **REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**

☐ **OTHER:** _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.